UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

ELIAS RICARDO VIEIRA

APLICAÇÃO PARA LOCALIZAÇÃO E GERENCIAMENTO DE SERVIÇOS DOMÉSTICOS PARA AUTÔNOMOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2019

ELIAS RICARDO VIEIRA

APLICAÇÃO PARA LOCALIZAÇÃO E GERENCIAMENTO DE SERVIÇOS DOMÉSTICOS PARA AUTÔNOMOS

Trabalho de Conclusão de Curso, apresentado ao Curso de Especialização Java, da Universidade Tecnológica Federal do Paraná, *Campus* Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Vinicius Pegorini

PATO BRANCO
2019

MINISTÉRIO DA EDUCAÇÃO
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Especialização em Tecnologia Java

TERMO DE APROVAÇÃO

APLICAÇÃO PARA LOCALIZAÇÃO E GERENCIAMENTO DE SERVIÇOS DOMÉSTICOS PARA AUTÔNOMOS

por

ELIAS RICARDO VIEIRA

Este trabalho de conclusão de curso foi apresentado em 26 de Junho de 2019, como requisito parcial para a obtenção do título de especialista em Tecnologia Java. Após a apresentação o candidato foi arguido pela banca examinadora composta pelos professores Vinícius Pegorini (orientador), Beatriz Terezinha Borsoi e Robison Cris Brito, membros de banca. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Vinícius Pegorini
Prof. Orientador (UTFPR)

_____
Beatriz Terezinha Borsoi
(UTFPR)

_____
Robison Cris Brito
(UTFPR)

_____
Robison Cris Brito
Coordenador do curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

Algumas pessoas que por não terem condições, sejam elas financeiras, de espaço físico, tempo ou disponibilidade, não possuem equipamentos necessários para lavar roupas e calçados. Contudo, existem pessoas que possuem esses equipamentos e tempo disponível para fazer tais serviços. Pensando nisso, neste trabalho foi desenvolvido um sistema web para localizar quem necessita com quem pode oferecer estes serviço, trazendo uma maneira para que o cliente possa, da forma que seja mais cômoda a ele, resolver a situação e não precisar adquirir um equipamento, além de possibilitar ao prestador do serviço o ganho de uma renda extra, por meio de um equipamento antes ocioso.

Palavras-chave: JavaScript. NodeJS. EmberJS. SailsJS. Serviços domésticos. Georreferenciamento.

ABSTRACT

Some people who lack financial, space, time or availability conditions to have the necessary equipment to wash clothes and shoes. However, there are people who have these equipment and time available to do such services. With this in mind, in this work a web system was developed to find out who needs these services, providing a way so that the client can, in a way that is more comfortable to him, to solve the situation and not need to acquire an equipment, besides to enable the service provider to earn an extra income through previously idle equipment.

Keywords: JavaScript. NodeJS. EmberJS. SailsJS. Domestic services. Georeferencing.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

API	*Application Programming Interface*
ACID	*Atomicity, Consistency, Isolation* e *Durability*
I/O	*Input/output*
CSS	*Cascading Style Sheet*
SPA	*Single Page Application*
REST	*Representational State Transfer*
CRUD	*Create, Retrieve, Update* e *Delete*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
RF	Requisito Funcional
RNF	Requisito Não Funcional
UTFPR	Universidade Tecnologia Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a introdução que é composta pelas considerações iniciais com o escopo e o contexto do trabalho, os objetivos e a justificativa. O texto é finalizado com a apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

Desde 2008, com o surgimento do Uber, empresas com modelo de negócio baseados em economia compartilhada, que é o compartilhamento e troca de serviços ou objetos, tem crescido e se tornado uma tendência, a ponto de especialistas afirmarem que em 2025 empresas com esse modelo de negócio irão movimentar mundialmente US$ 335 bilhões (DUNDER, 2018).

Empresas desse gênero, além de disponibilizarem produtos ou serviços de forma diferente da convencional, criam um impacto econômico e social relevante na vida dos fornecedores desses produtos ou serviços. “É uma pessoa com marca própria e que compartilha seu produto e seu serviço no mercado, a partir de uma empresa-mãe e por meio de novas tecnologias”, afirma Calvão, do Iniciativa Jovem (FONSECA, 2017).

Facilitar o acesso a essas ferramentas é um dos aspectos desse tipo de empresa, por isso, o desenvolvimento de aplicativos *web* é tão utilizado para esse setor. Segundo Borges (2016, p.s.n.) “o desenvolvimento de aplicativos Web não se limita apenas aos smartphones ou tablets: este tipo de aplicativo é projetado para rodar em qualquer navegador, seja em computadores fixos, laptops ou dispositivos móveis”.

Diante desse cenário verificou-se a possibilidade de desenvolver um aplicativo *web* que possa vincular pessoas que precisam do serviço de lavagem de roupas com pessoas dispostas a fornecer esse serviço.

1.2 OBJETIVOS

A seguir são apresentados o objetivo geral e os objetivos específicos deste trabalho.

1.2.1 Objetivo Geral

Desenvolver um aplicativo para localizar e solicitar serviços de lavagem de roupas para pessoas que possuam os equipamentos necessários para realizar esses serviços, sem a necessidade de ser uma empresa especializada na área.

1.2.2 Objetivos Específicos

- Direcionar pessoas que não tem acesso a máquinas de lavar com pessoas que estão disponíveis para prestar serviços desse gênero.
- Auxiliar os prestadores informais de serviço com uma renda extra.

1.3 JUSTIFICATIVA

Uma aplicação para localização e requisição de serviços domésticos voltados a lavagem de roupas auxiliará, tanto o solicitante, que poderá escolher um serviço de acordo com a localização e preço, quanto para o prestador do serviço, que poderá utilizar o tempo ocioso para gerar renda extra.

A sua implementação se justifica pela facilidade que trará aos utilizadores em encontrar pessoas disponíveis para prestarem o serviço desejado, considerando que o público alvo são pessoas que não possuem condições ou tempo para desenvolver essas tarefas e para os profissionais e prestadores de serviço utilizarem seus equipamentos que poderiam estar ociosos. Contudo, uma vez utilizados para a prestação de serviço de lavagem de roupa, passam a contribuir na renda familiar.

O sistema foi implementado para uso na *web*, facilitando assim, a localização de profissionais a partir do endereço atual do solicitante. Assim, o solicitante poderá escolher o prestador de serviço mais próximo da sua localização atual.

As tecnologias NodeJS, SailsJS e EmberJS foram escolhidas em decorrência dos recursos que elas oferecem para o desenvolvimento de aplicações *web*. Além da utilização de ferramentas do Google para localização geográfica, elas possibilitam uma forma visual da localização dos prestadores de serviço.

1.4 ESTRUTURA DO TRABALHO

Este texto está organizado em capítulos. O Capítulo 2 apresenta as ferramentas e as tecnologias utilizadas na modelagem e na implementação do sistema. No Capítulo 3 estão os resultados da realização do trabalho, representados por diagramas e complementações textuais, a apresentação do sistema desenvolvido por meio de suas telas e partes de código. Por fim estão as considerações finais, definidas por conclusão, e as referências utilizadas na composição do texto.

2 MATERIAIS E MÉTODO

Este capítulo apresenta as ferramentas e as tecnologias utilizadas na modelagem e na implementação do sistema. Também é apresentada a sequência das atividades desenvolvidas para a realização do trabalho.

2.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas para modelar e implementar o sistema.

Ferramenta/ Tecnologia	**Versão**	**Referência**	**Finalidade**
Lucidchart		https://www.lucidchart.com	Aplicativo para criação de diagramas
NodeJS	8.9.0	https://nodejs.org/	Linguagem de programação.
NPM	5.5.1	https://www.npmjs.com/	Gerenciador de pacotes para o NodeJS
WebStorm	2019.1	https://www.jetbrains.com/webstorm/	Ambiente de desenvolvimento.
PostgreSQL	11	https://www.postgresql.org	Banco de dados.
pgAdmin	4	https://www.pgadmin.org/	Administrador do banco de dados.
SailsJS	0.12.0	https://0.12.sailsjs.com/	Framework JavaScript para desenvolvimento backend
EmberJS	2.14.2	https://emberjs.com/	Framework JavaScript para desenvolvimento frontend
Bootstrap	4.0	https://getbootstrap.com/	Kit de ferramentas para desenvolvimento Web
Leaflet	1.3.4	https://leafletjs.com/	Biblioteca JavaScript para interação com mapas
SweetAlert2	7.0.5	https://limonte.github.io/sweetalert2/	Biblioteca JavaScript para criação de alertas.
geodesy	2.1.0	https://github.com/chrisveness/geodesy	Biblioteca JavaScript que auxilia na conversão de georreferenciamentos
moment.js	2.24.0	https://momentjs.com/	Biblioteca JavaScript para manipular data e hora
Google Maps	2.0.0	https://cloud.google.com/maps-platform/	APIs de geolocalização
jQuery Geocoding	1.6.5	https://github.com/ubilabs/geocomplete	Biblioteca JavaScript para autocompletar endereços

Quadro 1 – Ferramentas e tecnologias a serem utilizadas

A seguir são apresentadas, dentre as constantes no Quadro 1, as principais tecnologias utilizadas para o desenvolvimento da aplicação.

2.1.1 PostgreSQL

O PostgreSQL é um sistema de banco de dados objeto-relacional de código-fonte aberto que usa e estende a linguagem SQL combinada com diversos recursos que armazenam e dimensionam com segurança as cargas de trabalho de dados.

O PostgreSQL ganhou uma forte reputação por sua arquitetura comprovada, confiabilidade, integridade de dados, conjunto robusto de recursos, extensibilidade e dedicação da comunidade de código aberto por trás do software para fornecer soluções eficazes e inovadoras de maneira consistente. O PostgreSQL é executado em todos os principais sistemas operacionais, tem sido compatível com ACID desde 2001 e possui complementos importantes, como o popular extensor de banco de dados geoespacial PostGIS (POSTGRESQL, 2019).

2.1.2 NodeJS

O Node.js é um ambiente de execução para JavaScript, assíncrono e orientado a eventos de código aberto. O Node.js permite que os desenvolvedores usem JavaScript para escrever *scripts* do lado do servidor. O Node.js usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados por meio de dispositivos distribuídos (DEVMEDIA, 2019).

2.1.3 EmberJS

O Ember é um *framework front-end* em JavaScript projetado para ajudar a construir *websites* com interações de usuário ricas e complexas. Isso é feito fornecendo aos desenvolvedores os diversos recursos essenciais para gerenciar a complexidade em aplicativos da *Web* modernos, bem como um *kit* de ferramentas de desenvolvimento integrado que permite uma iteração rápida (EMBER, 2019).

2.1.4 SailsJS

O *framework* Sails foi desenvolvido por Mike McNeil para auxiliar sua equipe na criação de projetos escalonáveis do Node.js para clientes corporativos e iniciantes. Desde o seu lançamento em 2012, o Sails se tornou um dos *frameworks* de aplicações *web* mais utilizados no mundo (SAILSJS, 2019).

2.1.5 Google Maps API

Com o Google Maps é possível compartilhar detalhes sobre nomes de locais, endereços, classificações, comentários, dados de contato e ambiente. Os *Local Guides* e os usuários enviam milhões de atualizações todos os dias. Assim, é possível contar com informações precisas e confiáveis (GOOGLE CLOUD, 2019).

2.2 MÉTODO

O desenvolvimento se caracteriza como iterativo e incremental, embora com interações informais. A seguir as principais atividades realizadas para o levantamento e análise dos requisitos e a implementação da aplicação.

a) Requisitos - a definição dos requisitos foi realizada a partir da descrição dos requisitos do ponto de vista do usuário, realizada por possíveis clientes e profissionais prestadores de serviço.

b) Análise e projeto – na análise e projeto os requisitos foram modelados sob a forma de diagrama de casos de uso e de diagrama de entidades e relacionamentos do banco de dados. Revisões e complementos dos requisitos e sua respectiva modelagem ocorreram em diversos ciclos de implementação do sistema.

c) Desenvolvimento – inicialmente foram estudadas as tecnologias candidatas visando à identificação dos recursos que poderiam ser utilizados. Em seguida. a funcionalidade de cadastro de profissionais foi implementada, seguindo a busca de serviços, cadastro de clientes e cadastro de solicitações. As demais funcionalidades foram implementadas visando gerar executáveis funcionais.

d) Testes – os testes foram informais e com o objetivo de identificar erros de codificação.

3 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho, que é um sistema para localização e gerenciamento de pessoas que possuem roupas e afins a serem lavados, ligando-as com pessoas que possuem equipamentos necessários para realizar os procedimentos desejados.

3.1 ESCOPO DO SISTEMA

O aplicativo permitirá o cliente solicitar ao profissional do serviço desejado que os utensílios previamente informados sejam lavados. O profissional do serviço poderá disponibilizar quais os serviços prestados e aceitar a solicitação de clientes. Por exemplo: um profissional do serviço disponibiliza a função de lavar e secar, então, um cliente solicita a lavagem de uma camiseta e uma calça jeans, o profissional, por sua vez, aceita a solicitação e realiza os procedimentos necessários, atualizando o *status* do pedido para o acompanhamento do cliente.

As funcionalidades principais do aplicativo são:

a) Cadastro de usuários – profissionais e clientes.

b) Cadastro de serviços – utilizados para o profissional definir quais as atividades que irá oferecer.

c) Busca de serviços – utilizados para definir buscar os serviços desejados ordenado pelos mais próximos da localidade atual do cliente.

d) Cadastro de solicitações – o cliente faz uma solicitação de um serviço para um profissional.

e) *Status* do pedido – atualizado pelo profissional, o *status* do pedido é utilizado para que o cliente possa verificar a situação atual do pedido.

Cadastro de usuários – existem dois tipos de usuários, clientes e profissionais. Sendo que eles são diferenciados por um atributo chamado "*type"*. O cliente terá acesso aos pedidos que ele já realizou e o profissional terá acesso a todos os pedidos feitos a ele.

Cadastro de serviços – o usuário do tipo profissional deve informar os tipos de roupas pré-definidas que aceitará, os tipos são camisa, camiseta, calças, calções, vestidos, blusas, jaquetas, tênis, sapatos e roupas íntimas, deverá ser informando também o valor a ser cobrado por cada serviço.

Busca de serviços – o usuário do tipo cliente deve informar quais os tipos e a quantidade de roupas que deseja lavar, assim, o aplicativo filtrará os profissionais de serviços que aceitam o tipo selecionado, ordenando pela proximidade de endereços. Selecionando o profissional, ele decidirá se aceita ou não realizar o serviço.

Status do pedido – após aceitar uma solicitação de serviço, o profissional deve atualizar o *status* atual do pedido conforme o seu andamento. Ao finalizar, o cliente poderá informar uma nota para o serviço prestado, assim como o profissional poderá atribuir uma nota ao cliente.

3.2 MODELAGEM DO SISTEMA

Para solicitar um serviço, o cliente deve fazer uma busca, selecionar os serviços desejados e para finalizar o pedido, deverá registrar-se no sistema ou efetuar autenticação. O profissional, para ter acesso ao *dashboard*, também deverá estar autenticado. Para isso, ele poderá efetuar autenticação ou criar um novo usuário. A seguir os usuários e suas funções específicas:

a) **Profissional**

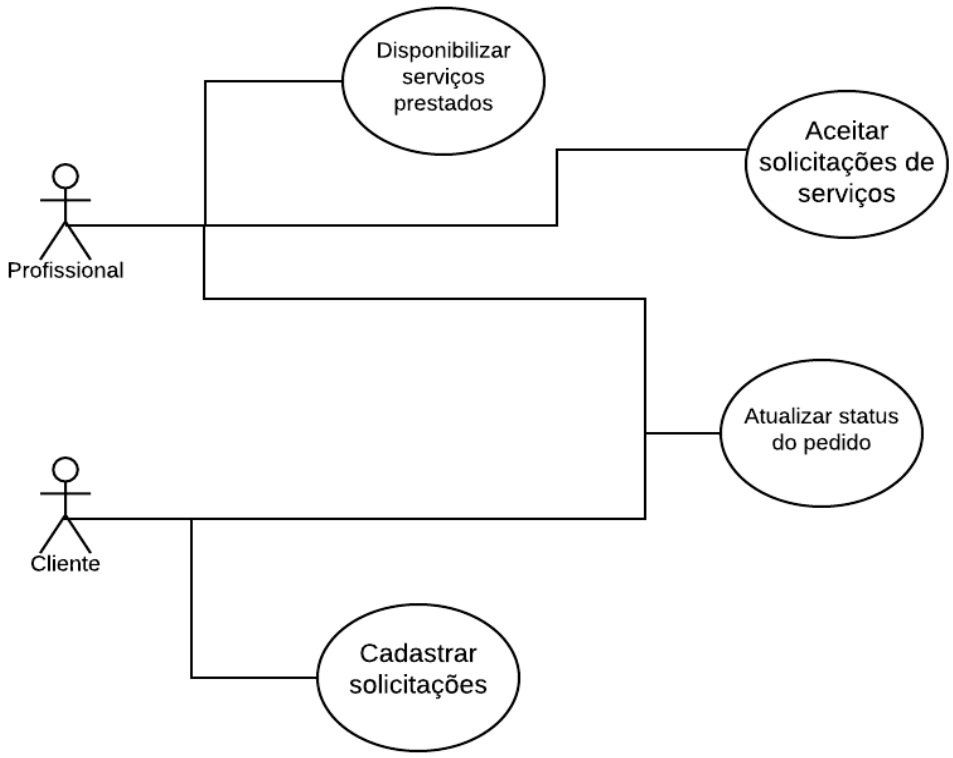
- Disponibilizar os serviços prestados.
- Aceitar solicitações de serviços.
- Atualizar status dos pedidos.
- Informar nota pelo relacionamento com o cliente.

b) Cliente

- Cadastrar solicitações.
- Informar nota pelo serviço prestado pelo profissional.

A Figura 1 apresenta o diagrama de casos de uso do sistema considerando os usuários como os dois atores do sistema.

Figura 1 – Diagrama de casos de uso



Fonte: autoria própria.

O Quadro 2 apresenta os requisitos funcionais agrupados por casos de uso apresentados na Figura 1.

Quadro 2 – Requisitos funcionais

	Caso de uso	Requisito	Descrição
RF01	Disponibilizar serviços prestados	Incluir serviço	Um usuário profissional pode incluir serviços.
RF03	Aceitar solicitações de serviços	Aceitar pedido de cliente	Um usuário profissional pode aceitar um pedido realizado por um cliente.
RF04	Aceitar solicitações de serviços	Recusar pedido de cliente	Um usuário profissional pode recusar um pedido realizado por um cliente.
RF05	Atualizar *status* do pedido	Mudar situação atual do pedido	Um usuário profissional pode atualizar o *status* do pedido, conforme ocorre seu andamento, informando o usuário cliente solicitante de cada alteração.
RF06	Atualizar *status* do pedido	Atribuir nota	Um usuário cliente pode atribuir uma nota pelo serviço prestado do fornecedor e um usuário fornecedor pode atribuir uma nota ao usuário cliente pela qualidade.
RF07	Cadastrar solicitações	Incluir solicitação	Um usuário cliente pode criar uma solicitação de serviço.

Fonte: autoria própria.

No Quadro 3 estão listados os requisitos não funcionais. Nesse quadro RNF significa Requisito Não Funcional.

Quadro 3 – Requisitos não funcionais

	Requisito	Descrição
RNF01	Acesso ao sistema	O cliente poderá realizar o login a partir do momento que selecionar os serviços que deseja e tentar prosseguir, caso já esteja autenticado, será possível prosseguir normalmente. O profissional, deverá acessar a tela para cadastro de profissional, que haverá o local possível para efetuar a autenticação.

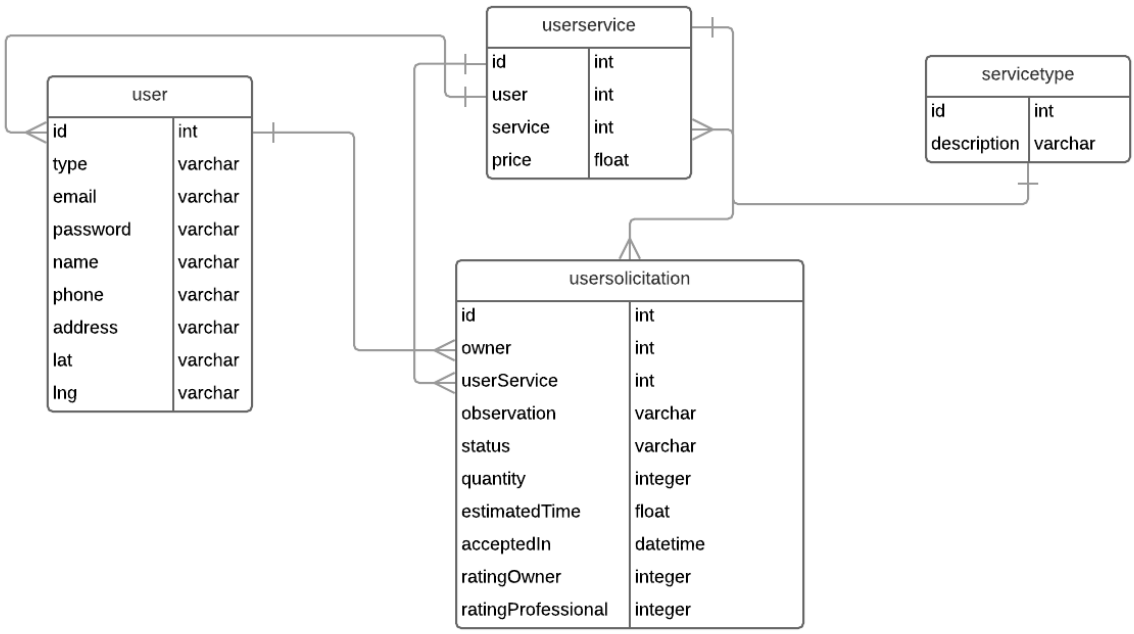
Fonte: autoria própria.

A Figura 2 apresenta o diagrama de entidades e relacionamentos do banco de dados. A tabela "user" armazena os dados dos profissionais e clientes. Cada tipo de usuário é diferenciado por meio do campo "type", o *login* será pelo campo "email".

A tabela "servicetype" armazena os serviços que os profissionais poderão utilizar. Relacionada a ela, a table "userservice" indica, precisamente, quais os serviços que o profissional disponibiliza e também o preço unitário de cada serviço.

Na tabela "usersolicitation" estarão os dados relacionados às solicitações dos clientes aos profissionais, contendo o serviço que o profissional presta, a quantidade deseja pelo cliente e a observação, caso necessária, para esse serviço. Nessa mesma tabela, o profissional deve atualizar o *status*, conforme o ocorrido. Ao finalizar, tanto o cliente como o profissional, deverão atribuir uma nota ao serviço, por meio do campo "ratingOwner" para o profissional avaliar o cliente e "ratingProfessional" para o cliente avaliar o profissional.

Figura 2 – Diagrama de entidades e relacionamentos



Fonte: autoria própria.

Os quadros a seguir apresentam a descrição das tabelas constantes na Figura 2.

O Quadro 4 apresenta a listagem dos campos da tabela de usuários.

Quadro 4 – Tabela user

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**
id	Numérico	Não	Sim	Não
type	Texto	Não	Não	Não
email	Texto	Não	Não	Não
Password	Texto	Não	Não	Não
name	Texto	Não	Não	Não
phone	Texto	Sim	Não	Não
address	Texto	Sim	Não	Não
lat	Numérico	Sim	Não	Não
lng	Numérico	Sim	Não	Não

Fonte: autoria própria.

No Quadro 5 está a descrição dos campos da tabela serviços.

Quadro 5 – Tabela service

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
id	Numérico	Não	Sim	Não
Description	Texto	Não	Não	Não

Fonte: autoria própria.

O registro dos serviços que os fornecedores prestam, é feita na tabela "userservice". O seu detalhamento está no Quadro 6.

Quadro 6 – Tabela userservice

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id	Numérico	Não	Sim	Não	
user	Numérico	Não	Não	Sim	Da tabela usuários
service	Numérico	Não	Não	Sim	Da tabela serviços
price	Numérico	Não	Não	Não	

Fonte: autoria própria.

O Quadro 7 apresenta a listagem dos campos da tabela de solicitações.

Quadro 7 – Tabela usersolicitation

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
id	Numérico	Não	Sim	Não	
owner	Numérico	Não	Não	Sim	Da tabela usuários
userService	Numérico	Não	Não	Sim	Da tabela "userservice"
observation	Texto	Não	Não	Não	
quantity	Numérico	Não	Não	Não	
status	Numérico	Não	Não	Não	
estimatedTime	Numérico	Não	Não	Não	
acceptedIn	Data	Não	Não	Não	
ratingOwner	Numérico	Não	Não	Não	
ratingProfessional	Numérico	Não	Não	Não	

Fonte: autoria própria.

3.3 APRESENTAÇÃO DO SISTEMA

O sistema é divido em três partes principais: a página inicial para busca de serviços, a página seleção de serviços e a parte administrativa, tanto do cliente como do profissional.

A Figura 3 apresenta a página inicial do sistema, que não necessita de autenticação. O conteúdo desta página é um painel apresentando os campos para realizar uma busca de serviços desejados e, também, apresenta um mapa com a localização atual do usuário e a localização dos profissionais cadastrados no sistema.

Figura 3 – Tela inicial

Fonte: autoria própria.

A Figura 4 mostra o conteúdo da página de solicitação de serviço. Nessa página o cliente verá todos os serviços, filtrando pelas opções selecionadas na tela anterior, ordenando de acordo com a distância atual para o endereço do profissional cadastrado. Também apresenta a lista de serviços selecionados para contratação, contendo o valor total solicitado, podendo adicionar uma observação para cada serviço selecionado. Clicando no *link* "Adicionar observação" será aberta uma janela *modal*, com o campo para adicionar a observação e se necessário, alterar a quantidade solicitada.

Figura 4 – Solicitação de serviços

Fonte: autoria própria.

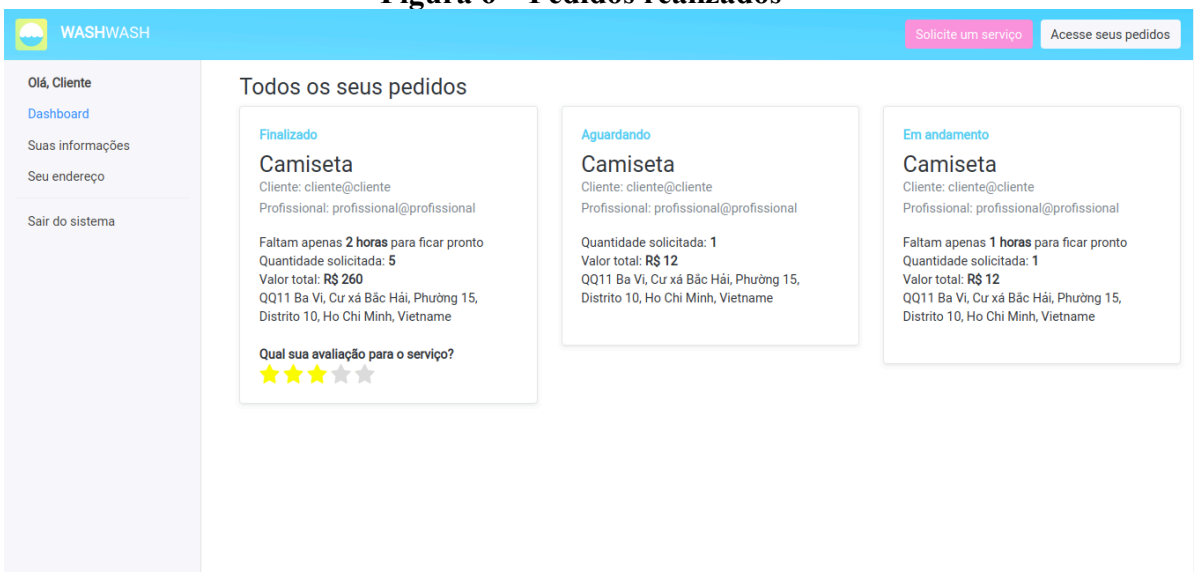
Clicando em Solicitar e continuar, presente na Figura 4, o sistema irá verificar se existe algum usuário autenticado. Se não existir, será apresentado um *popup* para o cliente informar o *email* e senha, se já possuir uma conta, ou criar uma nova conta, clicando no *link* "Ainda não possui conta? Clique aqui para criar", como pode ser observado na Figura 5.

Figura 5 – Autenticação de clientes

Fonte: autoria própria.

Após prosseguir, o cliente será redirecionado para a área administrativa, contendo todos os pedidos que ele fez, suas informações pessoais e o endereço cadastrado no sistema. Nessa página, ele poderá fazer as alterações nos dados pessoais como também, acompanhar o *status* dos projetos, além de poder avaliar o serviço recebido, após ser finalizado.

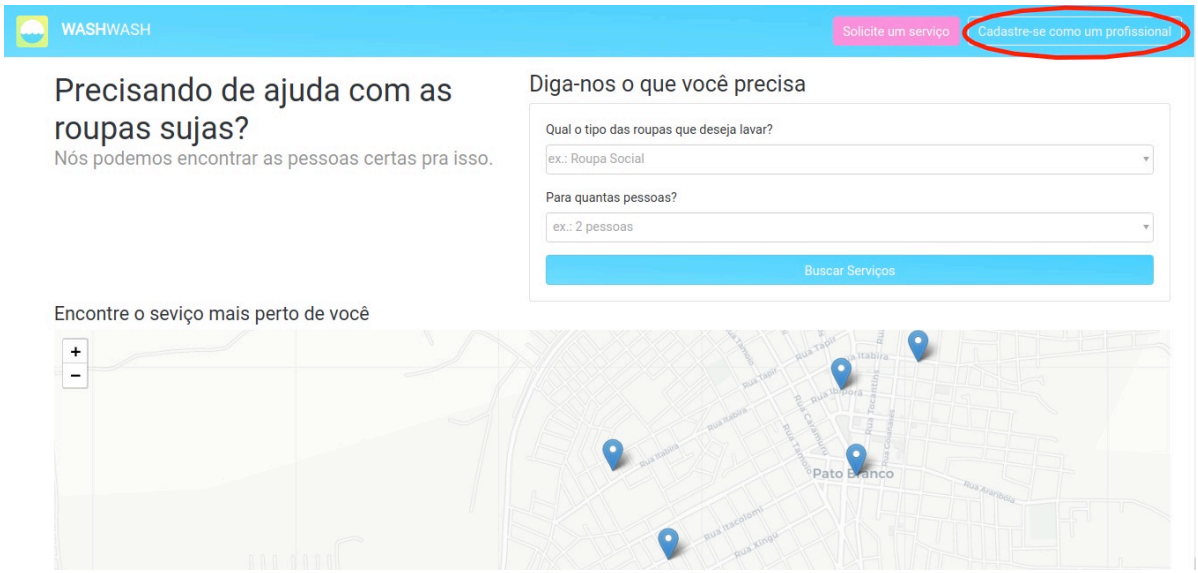
Figura 6 – Pedidos realizados



Fonte: autoria própria.

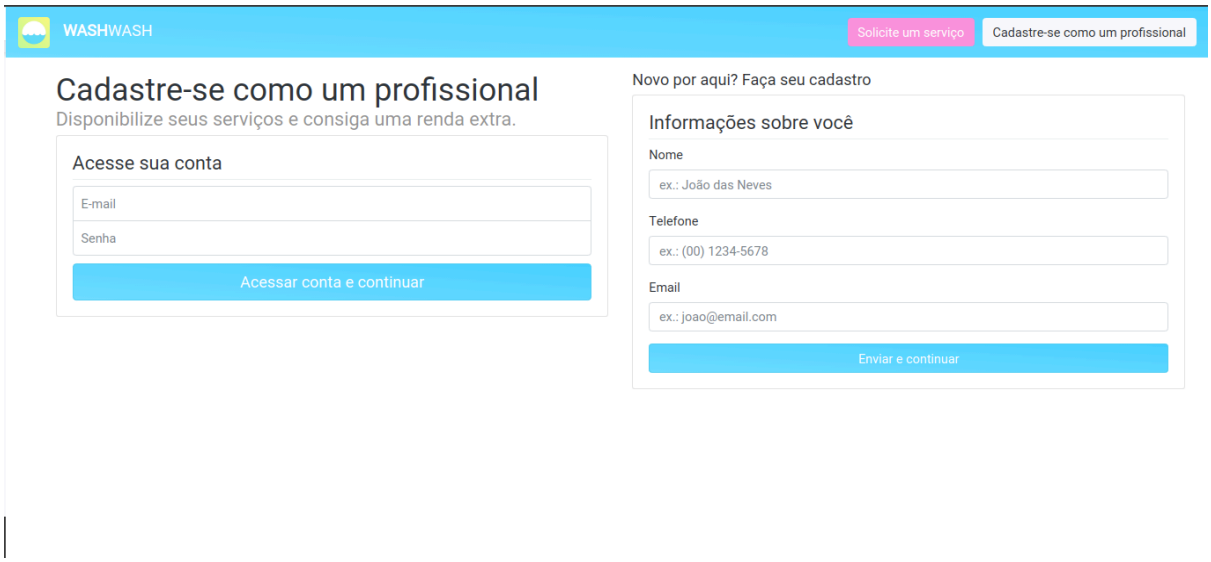
Para um profissional se cadastrar no sistema, basta clicar no botão “Cadastre-se como um profissional”, conforme pode ser visualizado na área circulada da Figura 7 e preencher as informações solicitadas. Caso o usuário já possua um cadastro, ele deve acessar a mesma tela e informar seus dados no painel “Acesse sua conta”, apresentado na Figura 8.

Figura 7 – Botão para cadastro de novos profissionais



Fonte: autoria própria.

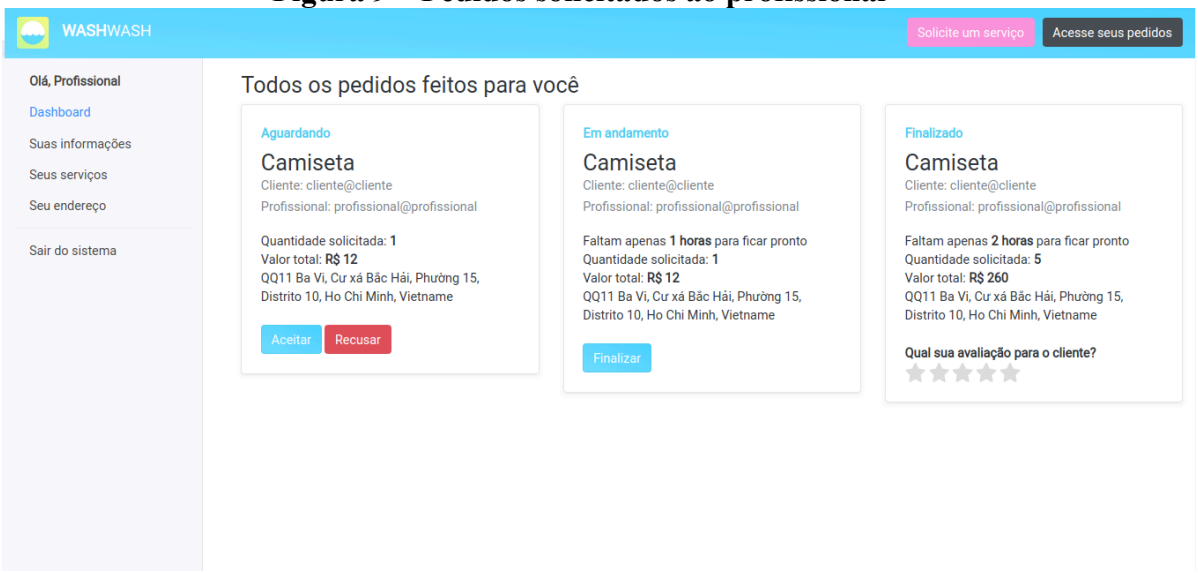
Figura 8 – Cadastro de profissionais



Fonte: autoria própria.

Após finalizar os processos para cadastrar um profissional ou acessar a conta já existente, o profissional terá acesso à página de administração dos pedidos feitos a ele. Nessa tela ele poderá aceitar ou recusar pedidos, sendo que, se clicar em aceitar, deverá informar o tempo que o serviço irá demorar para ficar pronto. Semelhante à página de administração do cliente, o profissional poderá alterar as informações pessoais e o endereço cadastrado no sistema, além disso, o profissional terá acesso a opção de adicionar os serviços que pretende disponibilizar aos clientes, como pode ser observado na Figura 9.

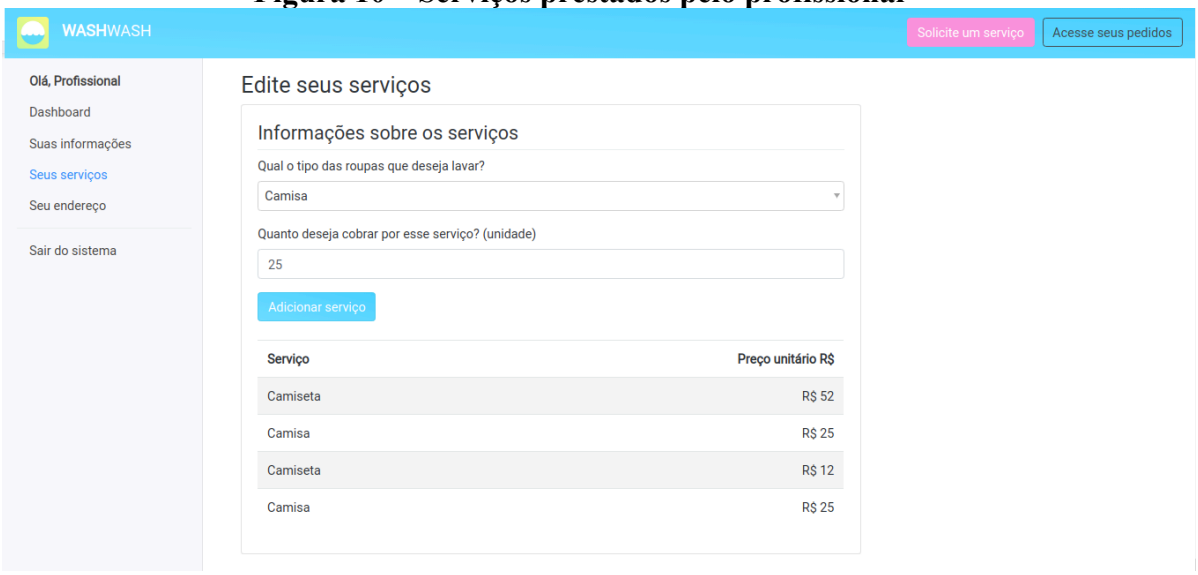
Figura 9 – Pedidos solicitados ao profissional



Fonte: autoria própria.

Na Figura 10 é mostrado como o profissional adiciona os serviços que irá prestar, selecionando tanto o serviço, como o preço por cada item, que será informado no momento da solicitação de serviços.

Figura 10 – Serviços prestados pelo profissional



Fonte: autoria própria.

3.4 IMPLEMENTAÇÃO DO SISTEMA

A implementação do aplicativo foi baseada no conceito de *Single Page Application* (SPA) e padrão de desenvolvimento REST utilizando JavaScript, EmberJS e SailsJS. SPA consiste basicamente em equilibrar e dividir o processamento de dados entre o servidor e o cliente. Dessa forma, o consumo de banda é menor, pois são menos dados trafegando e a experiência do usuário é mais fluida, uma vez que não é necessário recarregar a página para navegar entre as páginas.

A vantagem de utilizar *frameworks* como EmberJS e SailsJS é que, após instalados, é possível modelar o sistema apenas com linhas de comando executadas no terminal. Na Listagem 1 é apresentado como é feita a configuração e a conexão com o banco de dados.

Listagem 1 – Conexão com o banco de dados

```
module.exports.connections = {
 postgres: {
   adapter: 'sails-postgresql',
   host: 'localhost',
   user: 'postgres', // optional
   password: 'docker', // optional
   database: 'wash',
   port: '5434'//optional
 }
};
```

Fonte: autoria própria.

No arquivo "config/connections.js" é definido o nome da conexão e os dados, efetivamente, para conexão com o banco de dados. No arquivo "config/models.js" é setado a configuração criada na Listagem 1, além de definir o tipo da migração como "*alter"* que irá alterar a modelagem no banco de dados, toda vez que houver uma alteração e o servidor for iniciado, como pode ser visualizado na Listagem 2.

Listagem 2 –Definição da configuração utilizada

```
module.exports.models = {
 connection: 'postgres',
 migrate: 'alter'
};
```

Fonte: autoria própria.

Após essas configurações, é possível criar os *Controllers* e os *Models* no SailsJS, responsáveis por modelar o *backend* da aplicação, utilizando o comando apresentado na Listagem 3.

Listagem 3 – Comando responsável por criar a modelagem dos usuários

```
sails generate api user
```

Fonte: autoria própria.

O comando da Listagem 3 cria dois arquivos, um arquivo responsável por modelar o objeto e outro por gerar o *controller* já implementando todos os métodos *Create, Retrieve, Update* e *Delete* (CRUD*)* por padrão. Na Listagem 4 é apresentado o resultado da modelagem do objeto usuário.

Listagem 4 – Modelagem do objeto user

```
const bcrypt = require('bcrypt');

module.exports = {

 attributes: {
   email: {
     type: 'string',
   },
   password: {
     type: 'string',
   },
   type: {
     type: 'string',
   },
   name: {
     type: 'string',
   },
   phone: {
     type: 'string',
   },
   address: {
     type: 'string',
   },
   lat: {
     type: 'string',
   },
   lng: {
     type: 'string',
   },
   userServices: {
     collection: 'userService',
     via: 'user'
   },
```

```
 },
 beforeCreate(values, cb) {
   if (!values.password) {
     return cb({err: ["Password is required"]});
   }
   bcrypt.hash(values.password, 10, function (err, hash) {
     if (err) {
       return cb(err)
     }
     values.password = hash;

     cb();
   });
 },
 beforeUpdate(values, cb) {
   bcrypt.hash(values.password, 10, function (err, hash) {
     if (err) {
       return cb(err)
     }
     values.password = hash;
     cb();
   });
 }
};
```

Fonte: autoria própria.

Além de definir os atributos, na Listagem 4, são definidos os métodos *beforeUpdate* e *afterUpdate.* Esses métodos são utilizados para criptografar a senha do usuário. No controlador dos usuários, além dos métodos padrões, foi implementado o método para efetuar a autenticação, apresentado na Listagem 5.

Listagem 5 – Método de autenticação

```
login(req, res) {
 const email = req.body.email;
 if (!email || !req.body.password) {
   return res.badRequest({
     err: "Email or password cannot be empty"
   });
 }

 User.findOne({email: email}).exec(function (err, user) {
   if (err) {
     return res.serverError(err);
   }

   if (!user) {
     return res.notFound({err: 'Could not find email,' + email + '
sorry.'});
   }

   bcrypt.compare(req.body.password, user.password, function (err, result)
{
     if (result) {
       return res.json({
```

```
          user: user,
          token: jwToken.sign(user)
        });
      } else {
        return res.forbidden({err: 'Email and password combination do not
match'});
      }
    });
  });
},
```

Fonte: autoria própria.

No método de autenticação são enviados por meio de uma requisição POST os atributos *email* e *password*, o sistema localizará um usuário com *o e-mail* recebido por parâmetro e, se existir, comparará a senha recebida com a senha do usuário criptografada e armazenada no banco de dados. Assim como no SailsJS, o Ember JS também disponibiliza uma série de comandos para facilitar a criação da estrutura do aplicativo *frontend.*

Utilizando o comando apresentado na Listagem 6, o EmberJS se responsabiliza em criar uma estrutura para autenticação.

Listagem 6 – Comando responsável por criar a estrutura de autenticação

```
ember install ember-simple-auth-jwt
```

Fonte: autoria própria.

Após o término da execução desse comando, ficam disponibilizados no sistema, os métodos para configurar a autenticação. Na Listagem 7, é apresentado como o EmberJS se comunicará com o SailsJS.

Listagem 7 – app/pods/application/adapter.js

```
import DS from 'ember-data';
import environment from 'wash/config/environment';
import TokenAuthorizerMixin from 'ember-simple-auth-token/mixins/token-
authorizer';
import {computed} from '@ember/object';
import {inject as service} from '@ember/service';

export default DS.RESTAdapter.extend(TokenAuthorizerMixin, {
 session: service(),
 host: environment.urlServer,
 headers: computed('session.data.authenticated.token', function() {
   const headers = {};
   if (this.get('session.isAuthenticated')) {
     headers['Authorization'] = `Bearer
${this.get('session.data.authenticated.token')}`;
   }
```

```
    return headers;
  }),
});
```

Fonte: autoria própria.

Depois dessa configuração, é necessário informar ao EmberJS, que deve ser o endereço acessado para efetuar a autenticação, conforme apresentado na Listagem 8.

Listagem 8 – config/environment.js

```
'use strict';

module.exports = function(environment) {
  let ENV = {
    modulePrefix: 'wash',
    podModulePrefix: 'wash/pods',
    environment,
    //First address to access app
    rootURL: '/',
    locationType: 'auto',
    EmberENV: {
      EXTEND_PROTOTYPES: {
        Date: false
      }
    },

    APP: {

    }
  };

  if (environment === 'development') {
    //Address to API for execute locally
    ENV.urlServer = 'http://localhost:1337';
    //Address to authentication in API
    ENV.serverTokenEndpoint = `${ENV.urlServer}/users/login`;
    ENV.serverTokenRefreshEndpoint = `${ENV.urlServer}/users/check/`;
  }

  if (environment === 'production') {
    // here you can enable a production-specific feature
  }

  //authentication configuration
  ENV['ember-simple-auth-token'] = {
    // Header name added to each API request
    authorizationHeaderName: 'Authorization',
    // Prefix added to each API request
    authorizationPrefix: 'Bearer ',
    // Address for authentication in API
    serverTokenEndpoint: `${ENV.urlServer}/users/login`,
    // Address for refresh token in API
    serverTokenRefreshEndpoint: `${ENV.urlServer}/users/check/`,
    refreshTokenPropertyName: 'token',
    refreshAccessTokens: true,
```

```
    tokenExpirationInvalidateSession: false,
    // time to refresh token, in minutes
    refreshLeeway: 5
  };

  return ENV;
};
```

Fonte: autoria própria.

Nesse código são definidas as variáveis utilizadas em modo de desenvolvimento, para acesso local do aplicativo. Também é possível definir configurações necessárias para que ao compilar o aplicativo em modo de produção, acesse os endereços de, por exemplo, um servidor na nuvem. Além disso, é por meio desse código que toda a configuração do aplicativo *frontend* é definida. Exportando a variável "ENV" é possível ter acesso a todas essas definições, podendo também, adicionar conteúdos extras, como linguagem do sistema, endereço de acesso principal, entre outros.

Para o SailsJS comunicar-se corretamente com o padrão esperado pelo EmberJS, basta executar o comando apresentado na Listagem 9. Com ele, o Sails passará a utilizar o padrão *RESTAdapter,* um dos padrões aceitos pelo EmberJS, o que facilita na comunicação entre o *frontend* e *backend.*

Listagem 9 – Comando responsável por criar a estrutura de comunicação entre SailsJS e EmberJS

```
sails generate ember-blueprints
```

Fonte: autoria própria.

Para efetuar a autenticação entre o EmberJS e o SailsJS, depois das configurações criadas, basta implementar o código apresentado na Listagem 10, que utiliza o método *"autehenticate"* para enviar a requisição para o endereço definido e autenticar o usuário.

Listagem 10 – Método para envio dos dados para autenticação

```
import Component from '@ember/component';
import {inject as service} from '@ember/service';

export default Component.extend({
 session: service('session'),
 router: service(),
 actions: {
   async login() {
     await this.get('session').authenticate('authenticator:jwt', {
       email: this.get('email'),
       password: this.get('password')
     });
     const userId =
```

```
this.get('session.session.content.authenticated.user.id');
     const next = this.get('next');
     this.get('router').transitionTo(next, userId);
   }
 }
});
```

Fonte: autoria própria.

Para facilitar o desenvolvimento da interface gráfica, o EmberJS possibilita a criação de componentes. Na Listagem 11 é apresentado o componente que está vinculado com o código apresentado na Listagem 10.

Listagem 11 – Interface gráfica do formulário de autenticação

```
{{#if (not session.isAuthenticated)}}
 <div class="card card-login">
 <div class="card-body">
   <div class="form-signin">
     <h4 class="font-weight-normal">Acesse sua conta</h4>
     <div class="dropdown-divider"></div>
     <label for="inputEmail" class="sr-only">Email</label>
     {{input type="email" class="form-control" placeholder="E-mail"
required="true" value=email}}
     <label for="inputPassword" class="sr-only">Senha</label>
     {{input type="password" class="form-control" placeholder="Senha"
required="true" value=password}}
     <button class="btn btn-lg btn-primary btn-block" type="submit"
{{action "login"}}>Acessar conta e continuar</button>
   </div>
 </div>
</div>
{{/if}}
```

Fonte: autoria própria.

Esse código é responsável por criar a interface gráfica para efetuar a autenticação no sistema. Como é componentizado, é possível utilizar o mesmo componente para efetuar *login* tanto para usuário tipo cliente e profissional.

Na Listagem 12 é apresentado o código utilizado para o desenvolvimento do componente de mapa, apresentado na tela inicial do sistema.

Listagem 12 – Componente de mapa

```
import Component from '@ember/component';
import { inject as service } from '@ember/service';
import { A } from '@ember/array';

export default Component.extend({
 store: service(),
 geolocation: service(),
 lat: 0,
 lng: 0,
```

```
 zoom: 15,
 marks: A(),
 async didInsertElement() {
   let { coords } = await this.get('geolocation').getLocation();
   this.set('lat', coords.latitude);
   this.set('lng', coords.longitude);
   this.get('marks').pushObject(coords);
   const users = await this.get('store').query('user', {
     type: 'professional'
   });
   users.forEach(user => {
     this.get('marks').pushObject({
       latitude: user.get('lat'), longitude: user.get('lng')
     });
   })
 },
 actions: {
   updateCenter(e) {
     let center = e.target.getCenter();
     this.set('lat', center.lat);
     this.set('lng', center.lng);
   }
 }
});
```

Fonte: autoria própria.

Ao carregar a página, o componente é responsável por buscar todas as localizações dos profissionais cadastrados no sistema e a localização atual e inserir os marcadores no mapa. Na Listagem 13 apresenta como esse desenvolvimento é aplicado na interface gráfica.

Listagem 13 – Interface gráfica dos mapas

```
{{#leaflet-map lat=lat lng=lng zoom=zoom onMoveend=(action "updateCenter")
as |layers|}}
 {{tile-layer
url="https://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}.png"}}

 {{#each marks as |m|}}
   {{layers.marker lat=m.latitude lng=m.longitude}}
 {{/each}}

{{/leaflet-map}}
```

Fonte: autoria própria.

Para a apresentação gráfica dos mapas foi utilizada a biblioteca *leaflet-map*, que facilita na implementação e utilização de marcadores, além de disponibilizar gratuitamente uma opção de mapeamento.

As Listagens 14 e 15 mostram como é feito o cálculo entre a distância atual e a distância do profissional que presta o serviço filtrado na tela inicial. Sendo que na Listagem 14 está o código e na Listagem 15 a interface gráfica.

Listagem 14 – Código para cálculo da distância

```
import Component from '@ember/component';
import { observer } from '@ember/object';
import { inject as service } from '@ember/service';
import LatLon from 'geodesy/latlon-ellipsoidal-vincenty.js';

export default Component.extend({
 store: service(),
 geolocation: service(),
 distanceObserver: observer('service.user.{lat,lng}', async function() {
   const { coords } = await this.get('geolocation').getLocation();
   const p1 = new LatLon(coords.latitude, coords.longitude);
   const p2 = new LatLon(this.get('service.user.lat'),
this.get('service.user.lng'));
   const meters = p1.distanceTo(p2);
   const km = meters / 1000;
   const distance = km < 1 ? meters : km;
   this.set('unit', km < 1 ? 'm':'km');
   this.set('distance', (distance).toFixed(2));
   this.set('service.distance', this.get('distance'));
 }).on('didInsertElement'),
 actions: {
   add() {
     const userSolicitation = this.get('store').createRecord('user-
solicitation');
     userSolicitation.set('userService', this.get('service'));
     userSolicitation.set('quantity', this.get('model.quantity'));
     this.get('model.userSolicitations').pushObject(userSolicitation);
   }
 }
});
```

Fonte: autoria própria.

Listagem 15 – Interface gráfica dos resultados do filtro selecionado

```
<div class="card mb-4 shadow-sm">
 <div class="card-header">
   <h4 class="my-0 font-weight-normal">{{service.user.name}}</h4>
 </div>
 <div class="card-body">
   <h1 class="card-title pricing-card-title">
     R$ {{service.price}} <small class="text-muted">/ un.</small>
   </h1>
   <ul class="list-unstyled mt-3 mb-4">
     <li>Quantidade solicitada: <b>{{model.quantity}}</b></li>
     <li>Valor total: <b>R$ {{mult service.price model.quantity}}</b></li>
     <li>Lavagem de {{service.service.description}}</li>
     <li>{{service.user.address}}</li>
     <li>Distância de você <b>{{distance}} {{unit}}</b></li>
   </ul>
   <button {{action 'add'}} type="button" class="btn btn-lg btn-block {{if
(eq index 0) "btn-primary" "btn-outline-primary"}}">
     Contratar
   </button>
 </div>
</div>
```

Fonte: autoria própria.

Para buscar os pedidos solicitados aos profissionais é necessário realizar uma busca total dos serviços que o para profissional presta, filtrando pelo usuário autenticado. Na Listagem 16 é apresentada a busca em cada serviço que o usuário presta, sendo que existe uma lista das solicitações para este serviço.

Listagem 16 – Busca dos serviços solicitados ao profissional

```
import Route from '@ember/routing/route';
import {A} from '@ember/array';
export default Route.extend({
 async model(params, transition) {
   await this.store.findAll('user-solicitation');
   const user = transition.queryParams.user_id ||
transition.params["professional.dashboard"].user_id;
   const userServices = await this.store.query('user-service', { user });
   const userSolicitations = A([]);
   userServices.forEach(userService => {
     userService.get('userSolicitations').forEach(userSolicitation => {
       userSolicitations.pushObject(userSolicitation);
     });
   });
   return userSolicitations.sortBy('status', 'desc');
 },
});
```

Fonte: autoria própria.

Esta lista é apresentado no painel administrativo do profissional e possui o código mostrado na Listagem 17. Também nesse componente, o sistema verifica qual é a situação atual do pedido e dependendo da situação, o profissional poderá aceitar ou recusar, verificar se está em andamento e finalizar ou, após finalizado, atribuir uma nota ao cliente.

Listagem 17 – Componente utilizado na apresentação dos serviços solicitados

```
{{#solicitation-status solicitation=solicitation}}
 <div class="row">
   {{#if (eq solicitation.status 'Aguardando')}}
     <div class="col-sm-12">
       <a href="#" class="btn btn-primary" {{action 'status'
true}}>Aceitar</a>
       <a href="#" class="btn btn-danger" {{action 'status'
false}}>Recusar</a>
     </div>
   {{/if}}
   {{#if (eq solicitation.status 'Em andamento')}}
     <div class="col-sm-12">
       <a href="#" class="btn btn-primary" {{action
'finished'}}>Finalizar</a>
     </div>
   {{/if}}
   {{#if (eq solicitation.status 'Finalizado')}}
     <div class="col-sm-12">
```

```
        <b>Qual sua avaliação para o cliente?</b>
        {{star-rating rating=solicitation.ratingOwner wholeOnly=true
onClick=(action 'avaliation')}}
      </div>
    {{/if}}
 </div>
{{/solicitation-status}}
```

Fonte: autoria própria.

Listagem de 18 é apresentado o código responsável por efetuar as operações descritas na Listagem 17.

Listagem 18 – Código responsável por alterar o status dos pedidos, por parte do profissional

```
import Component from '@ember/component';
import moment from 'moment';
import Swal from 'sweetalert2';

export default Component.extend({
 tagName: '',
 rating: null,
 actions: {
   async status(value) {
     if (value) {
       this.set('solicitation.acceptedIn', moment().toDate());
       this.set('solicitation.status', 'Em andamento');
       const {value: estimatedTime} = await Swal.fire({
         title: 'Em quantas horas esse serviço ficará pronto?',
         input: 'number',
         showCancelButton: false,
         inputValidator: (value) => {
           if (!value) {
             return 'Você precisa informar o tempo estimado!'
           }
         }
       })

       this.set('solicitation.estimatedTime', estimatedTime);
     } else {
       this.set('solicitation.status', 'Não aceito');
     }
     await this.get('solicitation').save();
   },
   async finished() {
     this.set('solicitation.status', 'Finalizado');
     await this.get('solicitation').save();
   },
   async avaliation(rating) {
     this.set('solicitation.ratingOwner', rating);
     await this.get('solicitation').save();
   }
 }
});
```

Fonte: autoria própria.

O componente "*solicitation-status"* apresentado na Listagem 17 é utilizada tanto para apresentar os pedidos solicitados aos profissionais, como para apresentar os pedidos efetuados pelos clientes (Listagem 19).

Listagem 19 – solicitation-status

```
<div class="col-md-4">
 <div class="row no-gutters border rounded overflow-hidden flex-md-row mb-
4 shadow-sm h-md-250 position-relative">
   <div class="col p-4 d-flex flex-column position-static">
     <strong class="d-inline-block mb-2 text-
primary">{{solicitation.status}}</strong>
     <h3 class="mb-
0">{{solicitation.userService.service.description}}</h3>
     <div class="mb-1 text-muted">Cliente:
{{solicitation.owner.name}}</div>
     <div class="mb-1 text-muted">Profissional:
{{solicitation.userService.user.name}}</div>
     <ul class="list-unstyled mt-3 mb-4">
       {{#if solicitation.timeLeft}}
         <li>Faltam apenas <b>{{solicitation.timeLeft}} horas</b> para
ficar pronto</li>
       {{/if}}
       <li>Quantidade solicitada: <b>{{solicitation.quantity}}</b></li>
       <li>Valor total: <b>R$ {{mult solicitation.userService.price
solicitation.quantity}}</b></li>
       <li>{{solicitation.owner.address}}</li>
       <li>{{solicitation.observation}}</li>
     </ul>
     {{yield}}
   </div>
 </div>
</div>
```

Fonte: autoria própria.

4 CONCLUSÃO

A modelagem e a implementação de um sistema *web* para gerenciamento de pedidos e localização de prestadores de serviços autônomos foram realizadas como planejado, atendendo os objetivos definidos para o trabalho. Para o desenvolvimento foram utilizadas várias tecnologias, dentre elas a linguagem JavaScript que agregada a *frameworks* e componentes possibilita a implementação de aplicações *web* SPA.

Os objetivos propostos foram cumpridos, um aplicativo para localização e gerenciamento desses profissionais foi desenvolvido. O sistema permite aos clientes buscar e solicitar serviços e aos profissionais, aceitar, recusar e informar o *status* dos pedidos.

Em decorrência da quantidade e diversidade de tecnologias disponibilizadas para a implementação de aplicações, é notável que o desenvolvimento tornou-se muito mais

produtivo. A definição das tecnologias voltadas para JavaScript foram fundamentais para o desenvolvimento, considerando que todos os *frameworks* utilizados para o desenvolvimento desse aplicativo são baseados em JavaScript, mais especificamente em NodeJS, tornando todo o processo muito mais ágil. Além disso, as configurações necessárias para conexão com banco de dados na execução do sistema e no modo de desenvolvimento estão disponíveis nos próprios *frameworks*.

REFERÊNCIAS

BORGES, Francisco. **6 Diferentes tipos de desenvolvimento de aplicativos web**. Disponível em: <https://pt.yeeply.com/blog/tipos-desenvolvimento-de-aplicativos-web/>. Acesso em: 23 jun. 2019.

DEVMEDIA. **NodeJS**. Disponível em: <https://www.devmedia.com.br/nodejs/>. Acesso em: 19 jun. 2019.

DUNDER, Karla. **Economia compartilhada cresce em todo o mundo**. Disponível em: <https://noticias.r7.com/economia/economia-compartilhada-cresce-em-todo-o-mundo-26082018>. Acesso em: 23 jun. 2019.

EMBER. **EmberIS**. Disponível em: <https://guides.emberjs.com/release/getting-started/intro/>. Acesso em: 19 jun. 2019.

FONSECA, Mariana. **Afinal, por que tantos negócios copiam o modelo do Uber?** Disponível em: <https://exame.abril.com.br/pme/afinal-por-que-tantos-negocios-copiam-o-modelo-do-uber/>. Acesso em: 23 jun. 2019.

GOOGLE CLOUD, **Google Maps API**. Disponível em: <https://cloud.google.com/maps-platform/places/>. Acesso em: 19 jun. 2019.

POSTGRESQL, **PostgreSQL**. Disponível em: <https://www.postgresql.org/>. Acesso em: 19 jun. 2019.

SAILSJS, **SailsJS.** Disponível em: <https://0.12.sailsjs.com/>. Acesso em: 19 jun. 2019.